Parameter Setting for Multicore CMA-ES with Large Populations

Nacim Belkhir[1,2], Johann Dréo[1],
Pierre Savéant[1], and Marc Schoenauer[2]

[1]Thales Research & Technology, Palaiseau, France
[2]TAO, Inria Saclay Île-de-France, Orsay, France
{nacim.belkhir,johann.dreo,pierre.saveant}@thalesgroup.com
marc.schoenauer@inria.fr

Abstract. The goal of this paper is to investigate on the overall performance of CMA-ES, when dealing with a large number of cores — considering the direct mapping between cores and individuals — and to empirically find the best parameter strategies for a parallel machine. By considering the problem of parameter setting, we empirically determine a new strategy for CMA-ES, and we investigate whether Self-CMA-ES (a self-adaptive variant of CMA-ES) could be a viable alternative to CMA-ES when using parallel computers with a coarse-grained distribution of the fitness evaluations. According to a large population size, the resulting new strategy for Self-CMA-ES and CMA-ES, is experimentally validated on BBOB benchmark where it is shown to outperform a CMA-ES with default parameter strategy.

Keywords: Empirical Study, Numerical optimization, Metaheuristics, Algorithms Comparison

1 Introduction

Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [6] is one of the most efficient algorithms for real valued single-objective optimization problems. Thanks to its invariance properties [10], some default parameter values could be tuned using a rather small set of test functions [6], and nevertheless provide robust performances on a large variety of problems, from analytical benchmark functions [8] to many real-world applications (see, among many others, [7]).

With the end of Moore's years, increasing the speed of software nowadays requires an efficient parallelisation. Evolutionary Algorithms like CMA-ES can trivially be parallelized without modifying the underlying dynamics of the algorithm by distributing the computation of the fitnesses of the whole population on different slave nodes, the master node maintaining the population as a whole, and ensuring the reproduction phase. For optimal efficiency, the population size should be some multiple of the number of available computing units.

It turns out that the default value for the population size λ for CMA-ES is rather small, empirically set to $4+3\ln(n)$ [6], where n is the problem dimension.

And increasing λ without any further parameter tuning has been experimentally demonstrated to perform poorly for CMA-ES and other types of Evolution Strategies: [3] proposes a new update strategy for the global step-size; [21, 22] suggests to modify the ratio between number of parents and number of offspring. This paper investigates another approach to improve the performance of CMA-ES in a distributed setting: assuming some given number of cores, the use of computing resources is optimized by fixing the population size λ to this number of cores[1]. The goal is then to optimize the other parameters of CMA-ES to improve its performances.

Today, parameter tuning is acknowledged as a mandatory step toward efficient optimization algorithms at large [11], be they exact combinatorial optimization algorithms [12], or (possibly stochastic) heuristics and metaheuristics, among which Evolutionary Algorithms [5] (more in Section 2.1). Off-line tuning considers parameter tuning as a (meta-)optimization problem, and generic optimization algorithms can hence be applied [4, 12, 14, 18]. These methods have been in particular used to further improve CMA-ES performances [13, 15, 19], therefore suggesting that the same approach could be used to tackle the problem of a large λ – though leaving open the issue of the generality of such tuning [20].

On the other hand, optimization is a dynamic process, and the best parameter values at a given time of the search might no longer be efficient later. On-line parameter tuning therefore seems a very promising approach. However, there are very few (if any) examples of success of on-line tuning except in the history of Evolution Strategies, where CMA-ES, as its name suggests, is the most sophisticated of a long line of algorithms that do efficiently implement on-line adaptation of their main parameters. Yet, the adaptation mechanism of CMA-ES itself has some parameters, and a first approach to their on-line tuning has been recently proposed, leading to the so-called Self-CMA-ES [17], validated on a few test functions, and in the framework of a large population size.

The goal of this work is to investigate CMA-ES parameter tuning in a distributed context (fixed large λ), and in particular to compare experimentally the off-line and on-line approaches for different values of λ on the BBOB benchmark suite. The paper is organized as follows. Section 2 rapidly introduces the problem of parameter setting, and details the hyper-parameters of CMA-ES and how Self-CMA-ES adapts them. Section 3 introduces the experimental protocol that is used in Section 4.4 to validate some choices of Self-CMA-ES and compare the different approaches. Finally, the results are discussed and further research directions are proposed in Section 5.

2 State of the Art

2.1 Parameter Setting

It is today widely acknowledged that the performances of optimization algorithms are highly correlated with the values given to their parameters [11]. Fol-

[1] This also covers the case where λ is set to some multiple of the number of cores.

lowing the classification discussed in [5], one should distinguish between off-line and on-line parameter setting methods. In the off-line case (aka parameter tuning), the important secondary issue is that of the generality of the setting, and in the on-line case (aka parameter control), the distinction between dynamic, adaptive or self-adaptive approaches.

Off-line approaches view the problem of parameter tuning as an optimization problem in the space of parameters: the fitness of a parameter setting is the performance of the algorithm at hand, and any optimization method on the parameter space can be used given a practical way to compute the performance of the algorithm. Assuming that the user knows the quantity she/he is interested in (e.g., minimizing the runtime to reach a given solution quality, or optimizing the solution quality given a fixed computational budget), here comes into play the generality of the sought setting [20]. If the target of the experiments is a single (or a small number of) problem instance(s), the performance of the algorithm is computed by running it on each target instance (eventually aggregating over the different instances in the target set). But very often, the goal of parameter tuning is to find a robust setting that will give very good performances for some class of problem instances that cannot be enumerated. The performance of the algorithm is then approximated by running it on some carefully chosen test set of instances of the target class, hoping the result will be general enough to cover the whole class. Using large test sets improves the robustness of the setting, but increases the computational cost of the parameter setting process, as one single evaluation of the performance of a given parameter setting involves running the algorithm at hand once for all instances of the test set.

Several generic optimization methods have been adapted to handle parameter tuning and cope with the above-mentioned generalization issue, based on racing [16], on metaheuristics [18], on statistical modeling of the algorithm performance with Gaussian Processes [2], or on local search [14]. The most recent one, that has been used in this work, is SMAC (Sequential Model-based Algorithm Configuration)[2] [12], that uses random forest regression to model the algorithm performance as well as the uncertainty of its prediction. SMAC uses the Expected Improvement measure to choose, given a model, which parameter set to try next.

On-line parameter control, on the other hand, is concerned with tuning the parameter values during the run of the algorithm, thus avoiding any generalization issue and, more importantly, requiring little, if any, computational overhead. Three approaches should be distinguished [5], depending on how the parameters are modified during the run: in the deterministic approach, they are modified using a fixed schedule (that has to be designed off-line!); in the adaptive approach, the parameters are modified according to some feedback from the current state of the search; and in the self-adaptive approach, the parameters are subject to evolution: each individual (potential solution of the original optimization problem) carries its own parameters, and though selection applies only to the fitness,

[2] SMAC is freely available at `http://www.cs.ubc.ca/labs/beta/Projects/SMAC/`.

it is hoped that successive selections will only select individuals that carry good parameters.

Unfortunately, whilst adaptive or self-adaptive on-line control is potentially more efficient than off-line tuning, offering a way to adapt the parameters to the instance at hand, and to the current state of the search, there are very few examples of successful on-line control, and most of them are highly problem-dependent. As a matter of fact, the only success story of on-line parameter tuning is that of Evolution Strategies. A detailed presentation of the history of Evolution Strategies in this perspective is given in Section 3 of [5] and will not be repeated here due to space restrictions. We will directly switch to introducing CMA-ES, that can be viewed as the last link of the long chain of Evolution Strategies variants, that went from adaptive to self-adaptive and back to adaptive tuning of its Gaussian mutation.

2.2 CMA-ES

Let f be the real-valued objective function, defined on $\mathbb{R}^n$. CMA-ES [6] evolves a Gaussian distribution $\mathcal{N}\left(\boldsymbol{m}^t, (\sigma^t)^2\,\boldsymbol{C}^t\right)$ on $\mathbb{R}^n$ with mean $\boldsymbol{m}^t$ (the current estimate of the optimal solution) and covariance matrix $(\sigma^t)^2\,\boldsymbol{C}^t$, where the step-size σ^t is isolated from the covariance direction $\boldsymbol{C}$ so they can be adapted separately. The original $(\mu/\mu_w, \lambda)$-CMA-ES (Algorithm 1) works as follows. At iteration t, the current distribution $\mathcal{N}\left(\boldsymbol{m}^t, (\sigma^t)^2\,\boldsymbol{C}^t\right)$ is sampled, generating λ candidate solutions (line 5), whose fitness is computed (line 6). The new mean $\boldsymbol{m}^{t+1}$ is computed line 7 as the weighted sum of the best μ individuals according to f. The adaptation of the step-size σ^t is controlled by the evolution path $\boldsymbol{p}_\sigma^{t+1}$, that stores, with relaxation factor c_σ, the successive mutation steps $\frac{\boldsymbol{m}^{t+1}-\boldsymbol{m}^t}{\sigma^t}$ (line 8). The step-size is increased (resp. decreased) in the case of the length of the evolution path $\boldsymbol{p}_\sigma^{t+1}$ is longer (resp. smaller) than the expected length it would have under random selection (line 9). The covariance matrix is updated using both a rank-one update term, computing the evolution path $\boldsymbol{p}_c^{t+1}$ of successful moves of the mean $\frac{\boldsymbol{m}^{t+1}-\boldsymbol{m}^t}{\sigma^t}$ of the distribution in the original coordinate system (line 11) and the rank-μ update, a weighted sum of the covariances of successful steps of the best μ individuals (using the weights of the update of the mean – line 12). Two weights are used for this last update (line 13), c_1 for the rank-one term, and c_μ for the rank-μ term, hence c_1 and c_μ must be positive with $c_1 + c_\mu \leq 1$.

The default values of the parameters of the algorithm [6] are set in line 1, but are hidden to the user in the standard CMA-ES distributions, except for the population size λ and the number of selected parents μ. Though the already-mentioned invariance properties of CMA-ES [10] ensure some robutsness of the default setting, several improvements could be reached using off-line tuning of some of these parameters, namely λ (or more precisely the coefficient of λ as a function of n) and the ratio $\frac{\mu}{\lambda}$, as well as the parameters c_σ and d_σ for the adaptation of σ [13, 19]. Note that some additional parameters related to the stopping criterion are not presented in Algorithm 1, and have a large impact on the restart versions of CMA-ES [1]. These were also tuned using IRACE in [15].

Algorithm 1 The $(\mu/\mu_w, \lambda)$-CMA-ES (from [6])

1: **given** $n \in \mathbb{N}_+$, $\lambda = 4 + \lfloor 3 \ln n \rfloor$, $\mu = \lfloor \lambda/2 \rfloor$, $w_i = \frac{\ln(\mu+\frac{1}{2}) - \ln i}{\sum_{j=1}^{\mu} (\ln(\mu+\frac{1}{2}) - \ln j)}$ for $i = 1 \dots \mu$, $\mu_w = \frac{1}{\sum_{i=1}^{\mu} w_i^2}$, $c_\sigma = \frac{\mu_w + 2}{n + \mu_w + 3}$, $d_\sigma = 1 + c_\sigma + 2\max(0, \sqrt{\frac{\mu_w - 1}{n+1}} - 1)$, $c_c = \frac{4}{n+4}$, $c_1 = \frac{2}{(n+1.3)^2 + \mu_w}$, $c_\mu = \frac{2\,(\mu_w - 2 + 1/\mu_w)}{(n+2)^2 + \mu_w}$
2: **initialize** $\boldsymbol{m}^{t=0} \in \mathbb{R}^n, \sigma^{t=0} > 0, \boldsymbol{p}_\sigma^{t=0} = \mathbf{0}, \boldsymbol{p}_c^{t=0} = \mathbf{0}, \boldsymbol{C}^{t=0} = \mathbf{I}, t \leftarrow 0$
3: **repeat**
4: **for** $k = 1, \dots, \lambda$ **do**
5: $\boldsymbol{x}_k = \boldsymbol{m}^t + \sigma^t \mathcal{N}(\mathbf{0}, \boldsymbol{C}^t)$
6: $\boldsymbol{f}_k = f(\boldsymbol{x}_k)$
7: $\boldsymbol{m}^{t+1} = \sum_{i=1}^{\mu} \boldsymbol{w}_i \boldsymbol{x}_{i:\lambda}$
8: $\boldsymbol{p}_\sigma^{t+1} = (1 - c_\sigma)\boldsymbol{p}_\sigma^t + \sqrt{c_\sigma(2 - c_\sigma)}\sqrt{\mu_w}(\boldsymbol{C}^t)^{-\frac{1}{2}} \frac{\boldsymbol{m}^{t+1} - \boldsymbol{m}^t}{\sigma^t}$
9: $\sigma^{t+1} = \sigma^t \exp(\frac{c_\sigma}{d_\sigma}(\frac{\|\boldsymbol{p}_\sigma^{t+1}\|}{\mathbb{E}\|\mathcal{N}(\mathbf{0}, \mathbf{I})\|} - 1))$
10: $h_\sigma = \mathbb{1}_{\|p_\sigma^{t+1}\| < \sqrt{1-(1-c_\sigma)^{2(t+1)}}(1.4 + \frac{2}{n+1})\,\mathbb{E}\|\mathcal{N}(\mathbf{0}, \mathbf{I})\|}$
11: $\boldsymbol{p}_c^{t+1} = (1 - c_c)\boldsymbol{p}_c^t + h_\sigma \sqrt{c_c(2 - c_c)}\sqrt{\mu_w} \frac{\boldsymbol{m}^{t+1} - \boldsymbol{m}^t}{\sigma^t}$
12: $\boldsymbol{C}_\mu = \sum_{i=1}^{\mu} w_i \frac{\boldsymbol{x}_{i:\lambda} - \boldsymbol{m}^t}{\sigma^t} \times \frac{(\boldsymbol{x}_{i:\lambda} - \boldsymbol{m}^t)^T}{\sigma^t}$
13: $\boldsymbol{C}^{t+1} = (1 - c_1 - c_\mu)\boldsymbol{C}^t + c_1 \underbrace{\boldsymbol{p}_c^{t+1} \boldsymbol{p}_c^{t+1\,T}}_{\text{rank-one update}} + c_\mu \underbrace{\boldsymbol{C}_\mu}_{\text{rank}-\mu\text{ update}}$
14: $t = t + 1$
15: **until** *stopping criterion is met*

However, to the best of our knowledge, the parameter setting for the adaptation of the covariance matrix c_c (line 11, c_1 and c_μ (line 13) has only been addressed on-line in [17], and will now be detailed.

2.3 Self-CMA-ES

In Self-CMA-ES [17], the on-line tuning of c_c , c_1 , c_μ relies on the hypothesis that the best parameter configuration at time t is the one that would have maximized at time $t-1$ the likelihood of generating the best individuals selected at time t. At every iteration t, an auxiliary optimization algorithm (another CMA-ES, denoted CMA-ES$_{aux}$) is hence used to compute this optimal configuration. After computing the λ offspring at time t (lines 4-5 of Algorithm 1), the state of the algorithm at time $t-1$ is restored, and the optimization of parameters c_c , c_1 , c_μ proceeds as follows: for each triplet value (c_c , c_1 , c_μ), the virtual distribution parameters σ and $\boldsymbol{C}$ are computed (lines 8-13) from state $t-1$, and the performance of (c_c , c_1 , c_μ) is the likelihood of generating the best μ of the actual λ offspring at time t from this virtual distribution. The triplet (c_c , c_1 , c_μ) that maximizes this likelihood is returned and is then used, at time t, to complete the actual update of the actual mutation parameters of CMA-ES (lines 8-13).

A first issue is that computing the log-likelihood of generating μ given points of $\mathbb{R}^n$ from a given Gaussian is costly and numerically unstable. It was hence

replaced by a proxy, that works as follows. λ points are sampled from the virtual Gaussian, their virtual mean is computed (as in line 7), and the Mahalanobis distance between the actual μ best offspring at time t and this mean is computed. The sum of ranks of these distances used as a proxy for the likelihood. The detailed formal description of this proxy for the likelihood is given in [17], together with the global Self-CMA-ES algorithm.

A second issue is the possible overfitting of the parameters (c_c , c_1 , c_μ) due to a single and limited sampling of the actual offspring at time t. And a third issue is the computational cost of running a full CMA-ES$_{aux}$ inside every iteration of the master CMA-ES: even though no additional fitness computation of the main CMA-ES is required, and even though the dimension of the auxiliary optimization problem is only 3, sampling the virtual Gaussian distribution to evaluate the proxy likelihood of many triples (and here the dimension is n) has a non-negligible cost. However, both issues can be resolved simultaneously. First, the CMA-ES$_{aux}$ is not restarted from scratch at every iteration t of the main CMA-ES, but restarts from the state of the CMA-ES$_{aux}$ at the end of iteration $t-1$; Second, only a small number of iterations of CMA-ES$_{aux}$ is actually run, avoiding possible overfitting. Section 4.1 will describe some experimental validation of this procedure.

3 Experimental Setting

The remainder of the paper is devoted to presenting experimental comparison with the goal of validating some choices for Self-CMA-ES, and assessing when and how Self-CMA-ES is a better choice than CMA-ES with its default values.

BBOB testbench: All experiments use test functions from the Black Box Optimization Benchmark (BBOB)[3] [9]. BBOB testbench contains 24 functions, with known difficulty (e.g. non-separability, high conditioning, different levels of multi-modality, with or without global structure, etc) and for different dimensions (2, 3, 5, 10, 20, 40). BBOB also proposes an API for most programming languages. To avoid any bias, for each function, 15 trials are run, where for each trial, the optimum is moved and for the non-separable functions, the coordinate system is rotated. Foreach trial, a maximum number of function evaluations of $10^5 * n$ is given before the algorithm is killed. Only the noiseless versions of the functions were used here.

Performance Measure BBOB uses as performance measure the Expected Run Time, that counts the number of function evaluations used to reach a given target objective value. taking into account the runs that failed to reach that target value. This computational effort is normalized by dividing it by the dimension, when the results on different dimensions need to be aggregated. In this work, we only consider one target value 10^{-8}, and the number of function evaluations #FEs as a measure of comparison. However, because we are interested in the distributed performance, in a context where only the time-to-solution matters, we

[3] `http://coco.gforge.inria.fr`

propose a new performance measure, the **Virtual Wall Clock Time** ($VWCT$), focusing on the core usage, and formally defined as:

$$VWCT = \frac{\#\text{FEs}}{\lambda} = \frac{\#\text{FEs}}{\#\text{cores}} \tag{1}$$

The communication time is here neglected: in real situations on HPC clusters, it will be several orders of magnitude smaller than the computation time of the objective function (even if this is not true for BBOB functions).

Implementation: For all experiments, we used the Octave/MATLAB source code provided by authors of [17][4], that was modified in order to expose the parameters for automated parameter tuning, and/or to apply new parameter strategies to some parameters.

4 Experimental Results

Four series of experiments are conducted. A first goal is to validate some choices made in [17] for Self-CMA-ES; A second goal is to compare Self-CMA-ES with some off-line tuning of (c_c , c_1 , c_μ); A third goal is to identify the best strategy for the choice of μ; and the final goal is to assess on the whole BBOB benchmark suite, the performances of Self-CMA-ES with respect to CMA-ES (using the best setting that could be deduced from the previous experiments).

4.1 Validation of Self-CMA-ES

A first sanity check of Self-CMA-ES is performed by tuning the initial values of c_c , c_1 , c_μ with SMAC. The good news is that the performance of Self-CMA-ES is not sensitive to these initial values, as the adaptive mechanism takes over, whatever its initialization.

A second experiment checks the strategy for CMA-ES$_{aux}$ (see Section 2.3), running it for different number of iterations, or to full completion. The clear conclusion is that indeed, as argued in [17], the best results are obtained when running a single iteration of CMA-ES$_{aux}$ at each iteration of the main CMA-ES. Because of the space constraints, none of these validation experiments is detailed here.

4.2 On-line vs off-line tuning of c_c , c_1 , c_μ

In order to check the efficiency of the on-line tuning of c_c , c_1 , c_μ done by Self-CMA-ES, it should be compared to the off-line tuning of the same parameters (e.g., using SMAC, see Section 2.1) on the plain CMA-ES. However, because it was demonstrated in [3, 21] that the performance of CMA-ES (or other Evolution Strategies) with a large λ was highly dependent on μ and the adaptation of σ, and also because SMAC experiments are very costly, it was decided to run

[4] `https://sites.google.com/site/selfcmappsn/`

one single SMAC campaign, tuning μ and σ_0, the initial value for σ, for both algorithms (using the adaptation scheme advocated in [3, 21] is left for further work), and c_c , c_1 , c_μ for CMA-ES. Table 1 describes the experimental conditions. Note additionally that c_1 and c_μ must satisfy an additional constraint, that was handled by returning a very high fitness without running the algorithm when violated.

Test Functions	F1-Sphere, F8-Rosenbrock, F13-Sharp Ridge, F16-Rastrigin
Dimensions	10, 20
λ	λ_{def}, 50, 100, 200, 500, 1000, 1500, 2000
SMAC target for Self-CMA-ES	$\mu \in [1, \lambda]$, $\sigma_0 \in [0, 2]$
SMAC target for CMA-ES	$\mu \in [1, \lambda]$, $\sigma_0 \in [0, 2]$, $(c_c , c_1 , c_\mu) \in [0, 1]^3$

Table 1: Experimental setting for SMAC on CMA-ES and Self-CMA-ES.

Typical results are given in Figure 1. The best values for μ (Figure 1a) are in agreement with [21], i.e., are lower than the default $\frac{\lambda}{2}$. Some regularity with respect to λ could however be identified, and will be investigated in Section 4.3.

Figure 1b is typical of the behavior of the best values of σ_0. Apart the fact that they usually are lower than the value used in [17] (2.0), it was not possible to fit any relation with the dimention of the problem. However, the influence of this parameter seemed limited accross the experiments. Hence, all further experiments will use $\sigma = 1.3$, a rough average of all best values returned by SMAC.

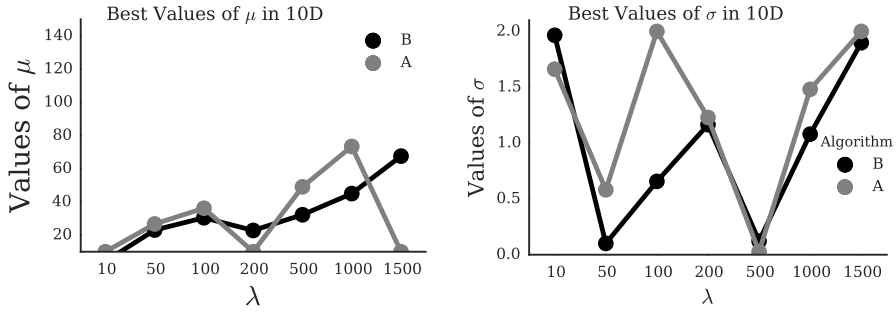
No trend could be observed either for c_c , c_1 , c_μ , except a rather large variance of the best values returned by SMAC. Thus the default parameter setting [6] will be used in the remaining of the experiments for CMA-ES.

Finally, Figure 1c plots the best overall values of both algorithms using the best parameterization returned by SMAC for each of them. The good news is that for all λ, Self-CMA-ES can be tuned to perform at least as good as the best tuning of CMA-ES, though the large variances suggest that more experiments should be run to better assess this conclusion.

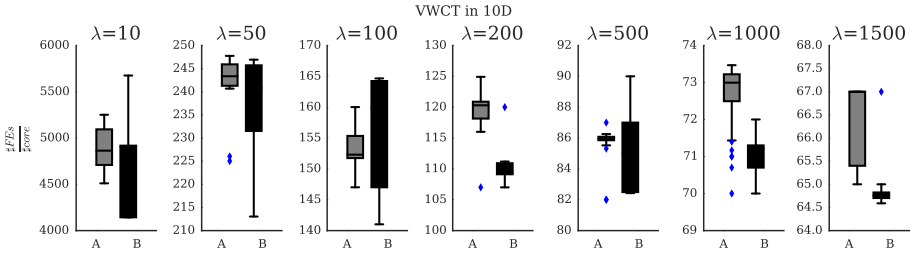
4.3 Choice of μ

The goal of the next series of experiments is to find a generic parametrization for Self-CMA-ES, i.e. a parametrization that is good on all instances without using SMAC for each new instance.The possible values for μ are hence restricted to the discrete list of values given on Table 2, depending on λ. As said, σ_0 is set to 1.3 and all other parameters are set to the default value. As for CMA-ES, the values for c_c , c_1 , c_μ are set to their default values as well – while they are of course adapted on-line by Self-CMA-ES.

Figure 2 displays the result for function F6-Attractive Sector in 10D for all (λ, μ) pairs. As for all functions of Table 2, the best values are obtained for



(a) Best Parameter Values of μ (b) Best Parameter Values of σ



(c) Best Performances

Fig. 1: Results for SMAC (see Table 1): Best values for μ (a) and σ_0 (b), and best performances (c) of CMA-ES (A) and Self-CMA-ES (B) on 10D-Rosenbrock.

$\mu \in [\frac{\lambda}{4}, \ln(\lambda)]$, while both algorithms achieve their worst performances whith the default strategy $\mu = \frac{\lambda}{2}$. The value $\mu = \frac{\lambda}{8}$ is hence retained for the final validation next Section, as providing quasi-optimal results for all functions.

Yet another validation of the on-line strategy for setting c_c , c_1 , c_μ is presented on Figure 3, that compares, for $\lambda = 200$, and on the F1-Sphere function on 10 and 20 dimensions, Self-CMA-ES with a CMA-ES for which c_c , c_1 , c_μ have been tuned using SMAC for each value of μ independently, denoted A* on the Figure. The results of the tuned CMA-ES are better than those of Self-CMA-ES, though not significantly for the chosen value $\mu = \frac{\lambda}{8}$. Furthermore, remember that the tuning with SMAC requires to run the algorithm several hundreds times. Furthermore, applying the parameters returned by SMAC for the 20D case to the 10D case displays results that are similar to those of Self-CMA-ES (not shown here).

Another interesting conclusion that can be drawn from Figure 2 is that the $VWCT$ for $\lambda = 500$ and $\lambda = 1000$ have very similar values: adding more cores does not help, and other strategies are needed to take full benefit of CMA-ES on large computing clusters.

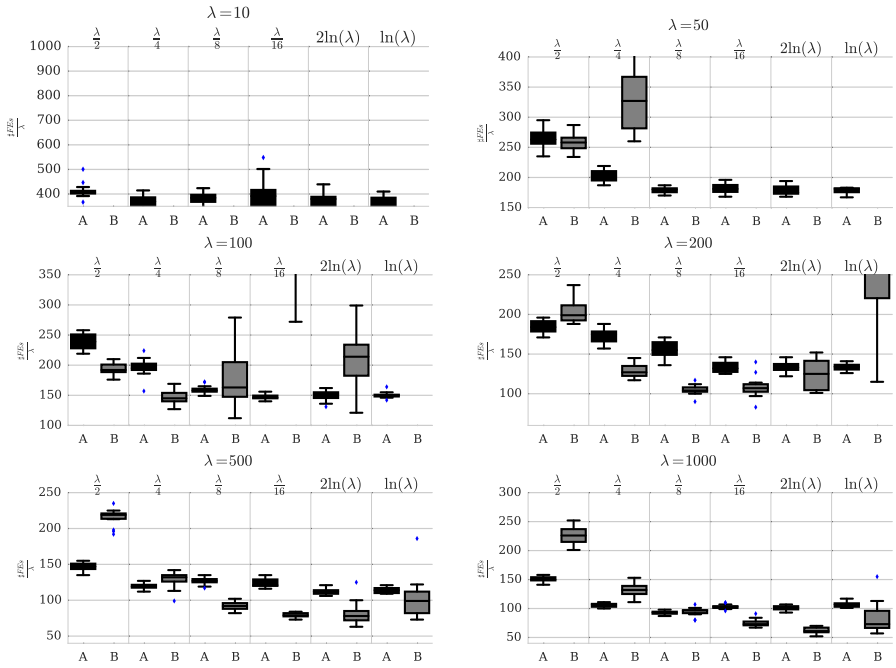


Fig. 2: Performances of CMA-ES (A, black) and Self-CMA-ES (B, grey) on 10D-Attractive Sector ($\lambda_{def} = 10$), for all pairs (λ, μ) of Table 2. Empty columns mean poor results (scaled for readability).

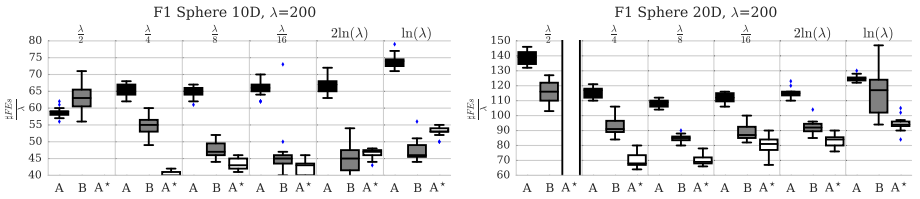
	Values
λ	(λ_{def}, 50, 100, 150, 200, 500, 1000)
μ	$\frac{\lambda}{2}$, $\frac{\lambda}{8}$, $\frac{\lambda}{16}$, $2 * ln(\lambda)$, $\ln(\lambda)$
Functions	F1-Sphere, F6-Attractive Sector, F8-Rosenbrock, F11-Discuss, F12-Bent Cigar
Dimensions	2, 10, 20

Table 2: Setting for the "μ" experiments. σ_0 is set to 1.3.

4.4 Overall BBOB Comparisons

The final experiment is to perform complete BBOB comparisons between the retained generic parametrization for both Self-CMA-ES and CMA-ES, i.e., $\mu = \frac{\lambda}{8}$ and $\sigma_0 = 1.3$ (the values of c_c , c_1 , c_μ are set to their default values for CMA-ES). The curve for the default strategy for CMA-ES ($\mu = \frac{\lambda}{8}$) was also added to the comparison. The case $\lambda = 200$ was chosen as representative.

Figure 4 displays the aggregated results for all functions, for dimensions 5, 10, 20 and 40. Except for dimension 5, Self-CMA-ES performs better than both CMA-ES, and the advantage increases with the dimension. Looking now at more detailed results, on Figure 5, it can be seen that the worst results for Self-CMA-



(a) (b)

Fig. 3: Comparison of CMA-ES (A, black), Self-CMA-ES (B, grey), and the tuned CMA-ES (A⋆, white) on the Sphere for $\lambda = 200$ and as in μ of Table 2.

ES are obtained for the separable functions in dimension 40 (Figure 5a), where it sometimes fails to reach the target value. Further investigations are needed to understand why this happens. Also note that the results for low or moderate conditioning functions (not shown here) show slightly worse results (though not as bad as for the separable functions) for Self-CMA-ES.

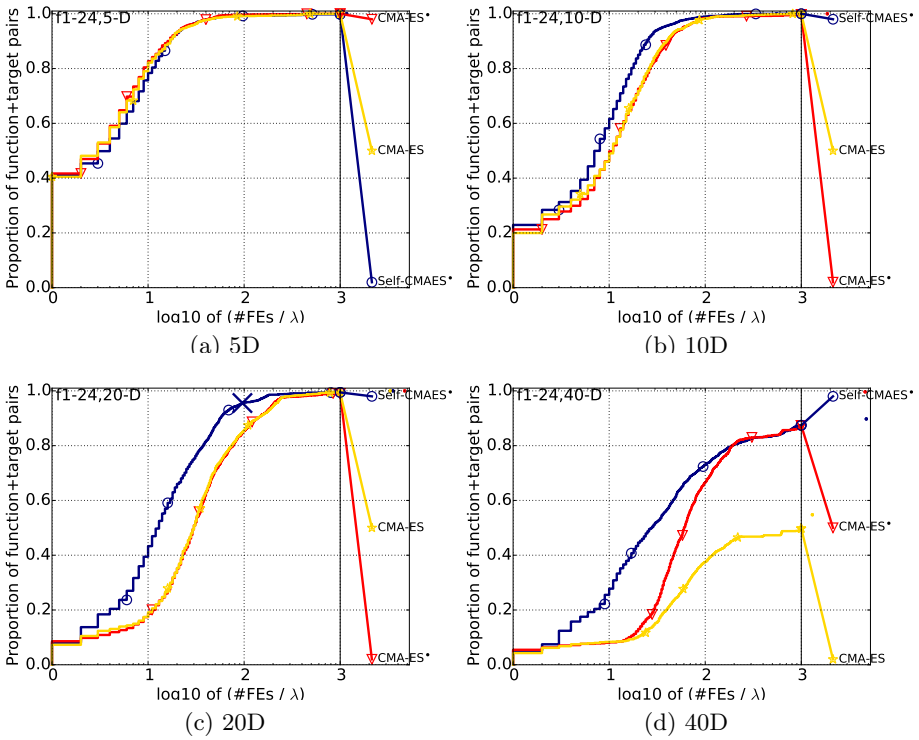
5 Discussion and Conclusion

This paper has experimentally studied parametrization strategies of CMA-ES that is run in a distributed environment when the primary goal is to minimize the wall-clock time-to-solution by using all available computing units (e.g., cores). This situation was simulated by considering large values of the population size λ as constraints, and tuning the other parameters accordingly. In particular, the Self-CMA-ES approach [17] has been demonstrated to be, in most cases, a viable alternative to the default CMA-ES for that goal.

The experiments presented in this paper have first validated most of the choices made in the original Self-CMA-ES approach [17] for the online control of the usually hidden parameters c_c , c_1 , c_μ that govern the update of the covariance matrix in CMA-ES. For values of λ up to 2000, we have observed that the best strategy for the choice of the number of parents μ is $\mu = \frac{\lambda}{8}$. This strategy outperforms the default strategy $\mu = \frac{\lambda}{2}$, for both CMA-ES and Self-CMA-ES. Also, this new strategy slightly outperforms the strategy $\mu = \frac{\lambda}{4}$ defined in [21], although [21] considers larger values of λ.

Regarding the initial value σ_0 for the step-size σ, the best value for both CMA-ES and Self-CMA-ES was found to be smaller than that used in [17] (σ=2), while nevertheless higher than the default value used [6] ($\sigma = 0.3$). The latter is explained by the increase of λ, resulting in a larger coverage of the search space by the initial sampling. Additionally, the new value of σ asserts the assumption of adapting the step-size when dealing with a larger λ, as proposed in [21].

The resulting new strategy for Self-CMA-ES and CMA-ES uncovers good performances, significantly outperforming the default strategy. Moreover, even



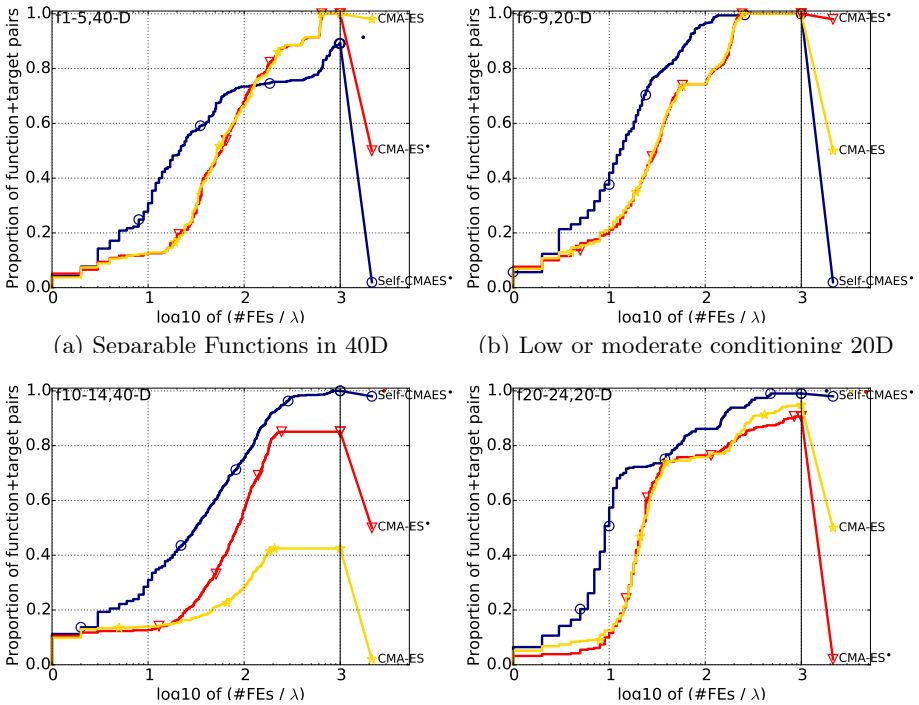
(a) 5D (b) 10D

(c) 20D (d) 40D

Fig. 4: Bootstrapped empirical cumulative distribution of $VWCT$ for all functions in 5D, 10D, 20D, 40D. Self-CMA-ES$^\bullet$ and CMA-ES$^\bullet$ use $\mu = \frac{\lambda}{8}$ while CMA-ES uses the default $\mu = \frac{\lambda}{2}$.

when CMA-ES is tuned off-line anew for each problem instance, Self-CMA-ES remains a good alternative to CMA-ES, performing only slightly worse while avoiding the huge computational cost of the tuning process.

More work is needed, however, in order to take full benefit of a very large number of computing units, as the wall-clock time performance seems to stagnate above 500 cores. Possible directions are to hybridize the method proposed here with those proposed in [3] and [21] and also modify the adaptation mechanism of the step-size σ. Another further direction is concerned with detecting situations where Self-CMA-ES adaptation mechanism performs poorly, and to switch back to the default values for c_c , c_1 , c_μ in such cases, thus guaranteeing performances at least as good as those of CMA-ES. Another approach would be to consider a portfolio of strategies in order to maximize the expected performance of CMA-ES, that should include CMSA-ES [3], that outperforms CMA-ES in large dimensions and population sizes.



(a) Separable Functions in 40D

(b) Low or moderate conditioning 20D

(c) high conditioning Functions in 40D

(d) Multi-modal functions with weak global structure in 20D

Fig. 5: Some Bootstrapped empirical cumulative distribution of $VWCT$. Legend as in Figure 4.

References

1. Auger, A., Hansen, N.: A Restart CMA Evolution Strategy with Increasing Population Size. In: CEC'05. vol. 2, pp. 1769–1776. IEEE (2005)
2. Bartz-Beielstein, T., Lasarczyk, C.W., Preuß, M.: Sequential Parameter Optimization. In: CEC'05. vol. 1, pp. 773–780. IEEE (2005)
3. Beyer, H.G., Sendhoff, B.: Covariance Matrix Adaptation Revisited–the CMSA Evolution Strategy–. In: G. Rudolph et al. (ed.) PPSN X, pp. 123–132. LNCS 5199, Springer Verlag (2008)
4. Birattari, M., Stützle, T., Paquete, L., Varrentrapp, K., et al.: A Racing Algorithm for Configuring Metaheuristics. In: William B. Langdon et al. (ed.) Proc. ACM GECCO'02. pp. 11–18 (2002)
5. Eiben, A., Michalewicz, Z., Schoenauer, M., Smith, J.E.: Parameter Control in Evolutionary Algorithms. In: Lobo, F., Lima, C.F., Michalewicz, Z. (eds.) Parameter Setting in Evolutionary Algorithms, pp. 19–46. Springer (2007)

6. Hansen, N., Müller, S., Koumoutsakos, P.: Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES). Evolution Computation 11(1), 1–18 (2003)
7. Hansen, N., Niederberger, S., Guzzella, L., Koumoutsakos, P.: A Method for Handling Uncertainty in Evolutionary Optimization with an Application to Feedback Control of Combustion. IEEE Transactions on Evolutionary Computation 13(1), 180–197 (2009)
8. Hansen, N.: Benchmarking a BI-population CMA-ES on the BBOB-2009 Function Testbed. In: Rothlauf, F. (ed.) GECCO Companion. pp. 2389–2396. ACM (2009)
9. Hansen, N., Auger, A., Finck, S., Ros, R.: Real-Parameter Black-Box Optimization Benchmarking 2010: Experimental Setup. Tech. Rep. RR-7215, INRIA (2010)
10. Hansen, N., Ros, R., Mauny, N., Schoenauer, M., Auger, A.: Impacts of Invariance in Search: When CMA-ES and PSO Face Ill-Conditioned and Non-Separable Problems. Applied Soft Computing 11, 5755–5769 (2011)
11. Hoos, H.H.: Programming by Optimization. Communications of the ACM 55(2), 70–80 (2012)
12. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential Model-Based Optimization for General Algorithm Configuration. In: Coello, C.A.C. (ed.) Learning and Intelligent Optimization. pp. 507–523. LNCS 6683, Springer Verlag (2011)
13. Hutter, F., Hoos, H.H., Leyton-Brown, K., Murphy, K.P.: An experimental investigation of model-based parameter optimisation: SPO and beyond. In: Rothlauf, F. (ed.) GECCO'09. pp. 271–278. ACM (2009)
14. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ParamILS: an Automatic Algorithm Configuration Framework. JAIR 36(1), 267–306 (2009)
15. Liao, T., Stützle, T.: Testing the impact of parameter tuning on a variant of IPOP-CMA-ES with a bounded maximum population size on the noiseless BBOB testbed. In: Proc. ACM GECCO. pp. 1169–1176. ACM (2013)
16. López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., Birattari, M.: The R-package Irace, Iterated Race for Automatic Algorithm Configuration. Tech. Rep. TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium (2011)
17. Loshchilov, I., Schoenauer, M., Sebag, M., Hansen, N.: Maximum Likelihood-based Online Adaptation of Hyper-parameters in CMA-ES. In: Thomas Bartz-Beielstein et al. (ed.) PPSN XIII, pp. 70–79. LNCS 8672, Springer Verlag (2014)
18. Nannen, V., Eiben, A.E.: Relevance Estimation and Value Calibration of Evolutionary Algorithm Parameters. In: IJCAI'07. vol. 7, pp. 6–12 (2007)
19. Smit, S., Eiben, A.: Beating the "World champion" Evolutionary Algorithm via REVAC Tuning. In: Proc. IEEE Congress on Evolutionary Computation. pp. 1–8 (July 2010)
20. Smit, S., Eiben, A.: Parameter Tuning of Evolutionary Algorithms: Generalist vs. Specialist. In: Cecilia Di Chio et al. (ed.) Applications of Evolutionary Computation. pp. 542–551. LNCS 6024, Springer Verlag (2010)
21. Teytaud, F.: A new selection ratio for large population sizes. In: Cecilia Di Chio et al. (ed.) Applications of Evolutionary Computation, pp. 452–460. LNCS 6024, Springer Verlag (2010)
22. Teytaud, F., Teytaud, O.: Log (λ) modifications for optimal parallelism. In: Robert Schaefer et al. (ed.) PPSN XI, pp. 254–263. LNCS 6238, Springer Verlag (2010)